How to Troubleshoot Log Rotation Issue

One of the problems, when comes to the log rotation, is that the log file does not seem rotated as the configuration file states. It usually ends up with one big log file. The expected log file should be relatively smaller. In this article, I am going to walk you through log rotation.

If we have a production-ready Django application, there are mainly three kinds of log rotation that we would be interested in. One is the application log. The second is the server log. Depending on the server, it could be uWSGI, Gunicorn, etc. In this article, I am going to use uWSGI as an example. The third one is the logrotate utility which is installed by default on Ubuntu.

Sample Project

In this article, I am going to use a sample project to demonstrate. I uploaded it on my Github. If you are interested, please check out the following Git repository.

https://github.com/slow999/DjangoAndLogrotate

Log Rotation 101

How does log rotation work? There are a few ways. The most common way is that when a rotation is triggered, it deletes the oldest copy, renames the rest of the copies along with the current one by the next sequence number, and creates a new log file. For example, the current log file is **mysite.log**. After the first rotation, the list of files would be **mysite.log** and **mysite.log.1**. After the second rotation, the list of files would be **mysite.log, mysite.log.1** and **mysite.log.2**. Let's say we make the number of copies being 2. Thus, after the third rotation, the list of files would still be **mysite.log, mysite.log.1,** and **mysite.log.2**.

Here we should know **inode** a little bit. As the log rotation runs, a new **mysite.log** is created and associated with a new inode. If a file is deleted, its inode won't recycle. Therefore, we can use inode to verify if log rotation is running properly. Next, I am going to walk you through these three types of log rotation using our sample project.

Django/Python Log Rotation

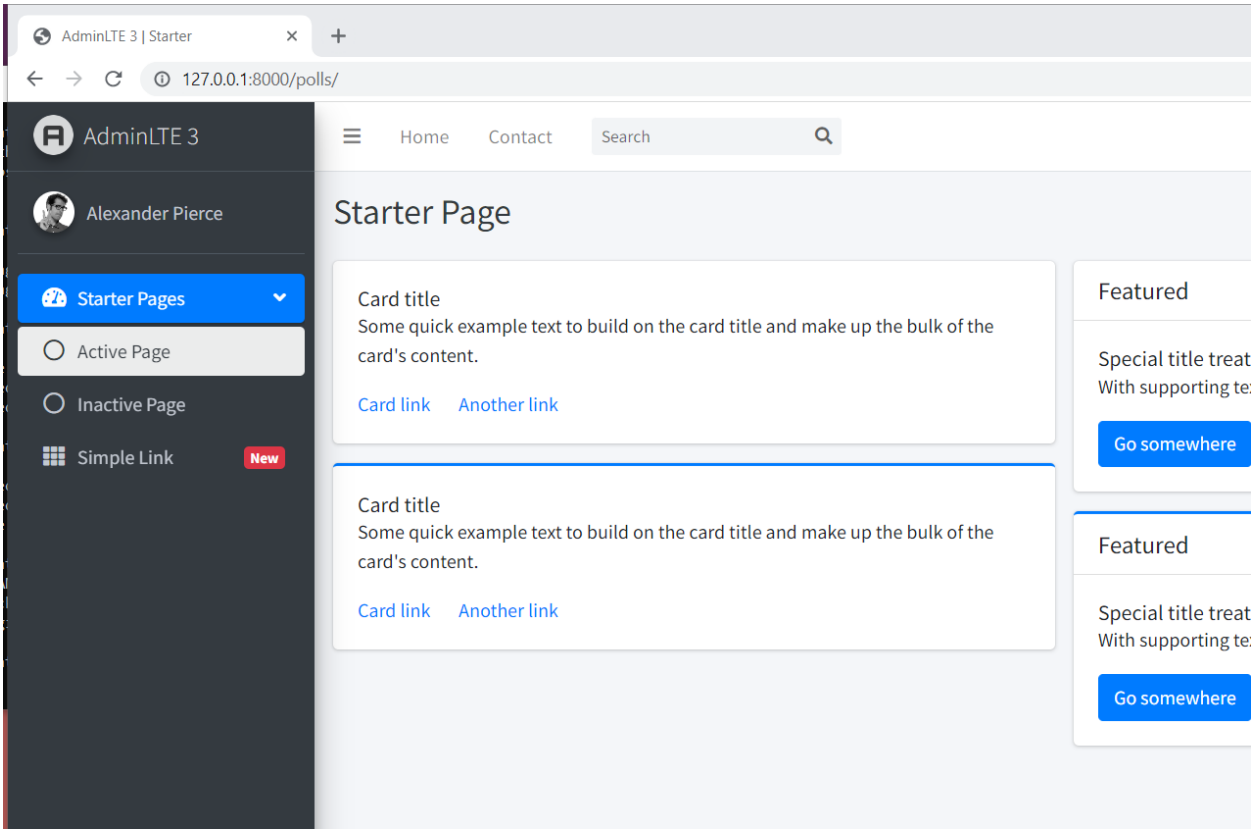
To launch the project, type the command as follows:

```
docker compose up -d
```

We would expect the output in the console as follows:

```
C:\Users\slow9\Documents\MyGit\DjangoAndLogrotate>docker compose up -d
[+] Running 3/3
 - Network djangoandlogrotate_default    Created                                0.9s
 - Container djangoandlogrotate-web-1    Started                                1.3s
 - Container djangoandlogrotate-nginx-1  Starte...                              2.4s
```

Open http://127.0.0.1:8000/polls/ in the browser. We would expect the page as below. It means our project is running.



To display running containers, type the command as follows:

docker ps

We would expect the output as below. Let's take down the container id that is highlighted in yellow. In my case, it's 0d56a6ed21f0.

```
C:\Users\slow9\Documents\MyGit\DjangoAndLogrotate>docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS          PORTS                    NAMES
6d2c33b960a2   djangoandlogrotate_nginx   "/docker-entrypoint.…"   2 minutes ago    Up 2 minutes    0.0.0.0:8000->80/tcp     djangoandlogrotate-nginx-1
0d56a6ed21f0   djangoandlogrotate_web     "uwsgi --ini mysite.…"   2 minutes ago    Up 2 minutes    8000/tcp                 djangoandlogrotate-web-1
```

To get into a running container, type the command as follows:

docker exec -it 0d56a6ed21f0 sh

We would expect the following output. It means we are in the container.

```
C:\Users\slow9\Documents\MyGit\DjangoAndLogrotate>docker exec -it 0d56a6ed21f0 sh
/code #
```

Let's list what is inside of **code** directory by typing the command as follows:

ls -li

We would expect the following output. Please note that the first column represents the inode. The inode of **mysite.log** is 43132. The **mysite.log** is generated by the Django application.

```
/code # ls -li
total 40
  43079 -rwxr-xr-x    1 root     root           674 Sep 26 04:04 Dockerfile
  43069 -rwxr-xr-x    1 root     root           647 Apr  3 15:46 manage.py
  43114 drwxr-xr-x    1 root     root          4096 Sep 26 04:47 mysite
  43132 -rw-r--r--    1 root     root           919 Sep 29 03:59 mysite.log
  43089 -rwxr-xr-x    1 root     root           233 Sep 26 04:03 mysite.uwsgi.ini
  43133 drwxr-xr-x    1 root     root          4096 Sep 26 04:47 polls
  32999 -rwxr-xr-x    1 root     root            14 Apr  3 15:46 requirements.txt
  42935 drwxr-xr-x    4 root     root          4096 Sep 26 04:47 static
```

The configuration of application logging is set up in the **mysite/settings.py** file. The portion of log rotation is as below. It means if the file size exceeds 1KB, it would invoke log rotation. Django would keep up to 2 copies of log files.

```
logging.config.dictConfig({
    …
    'handlers': {
        'logfile': {
            'level': 'DEBUG',
            'class': 'logging.handlers.RotatingFileHandler',
            'filename': 'mysite.log',
            'maxBytes':  1024 * 1,  # 1KB
            'backupCount': 2,
            'formatter': 'standard',
        },
        'console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
            'formatter': 'standard',
        },
    },
    …
})
```

Let's refresh the http://127.0.0.1:8000/polls/ page one time. List the **code** directory again. We should expect the following outputs. According to the inode, we know that the file whose inode is 43132 has been renamed from **mysite.log** to **mysite.log.1**. A new log file whose inode is 42870 has been created. In addition, please note that the modified time of both log files is the same.

```
/code # ls -li
total 44
  43079 -rwxr-xr-x    1 root     root           674 Sep 26 04:04 Dockerfile
  43069 -rwxr-xr-x    1 root     root           647 Apr  3 15:46 manage.py
  43114 drwxr-xr-x    1 root     root          4096 Sep 26 04:47 mysite
  42870 -rw-r--r--    1 root     root           856 Sep 29 04:13 mysite.log
  43132 -rw-r--r--    1 root     root           982 Sep 29 04:13 mysite.log.1
  43089 -rwxr-xr-x    1 root     root           233 Sep 26 04:03 mysite.uwsgi.ini
  43133 drwxr-xr-x    1 root     root          4096 Sep 26 04:47 polls
  32999 -rwxr-xr-x    1 root     root            14 Apr  3 15:46 requirements.txt
  42935 drwxr-xr-x    4 root     root          4096 Sep 26 04:47 static
```

Let's refresh the http://127.0.0.1:8000/polls/ page again. Keep watching the changes of the log files. The output of the third refresh is as follows:

```
/code # ls -li
total 48
  43079 -rwxr-xr-x    1 root     root           674 Sep 26 04:04 Dockerfile
  43069 -rwxr-xr-x    1 root     root           647 Apr  3 15:46 manage.py
  43114 drwxr-xr-x    1 root     root          4096 Sep 26 04:47 mysite
  42872 -rw-r--r--    1 root     root           856 Sep 29 04:27 mysite.log
  42870 -rw-r--r--    1 root     root           919 Sep 29 04:27 mysite.log.1
  43132 -rw-r--r--    1 root     root           982 Sep 29 04:13 mysite.log.2
  43089 -rwxr-xr-x    1 root     root           233 Sep 26 04:03 mysite.uwsgi.ini
  43133 drwxr-xr-x    1 root     root          4096 Sep 26 04:47 polls
  32999 -rwxr-xr-x    1 root     root            14 Apr  3 15:46 requirements.txt
  42935 drwxr-xr-x    4 root     root          4096 Sep 26 04:47 static
```

The output of the fourth refresh is as below. The log file whose inode is 43132 is deleted.

```
/code # ls -li
total 48
  43079 -rwxr-xr-x    1 root     root           674 Sep 26 04:04 Dockerfile
  43069 -rwxr-xr-x    1 root     root           647 Apr  3 15:46 manage.py
  43114 drwxr-xr-x    1 root     root          4096 Sep 26 04:47 mysite
  42873 -rw-r--r--    1 root     root           856 Sep 29 04:28 mysite.log
  42872 -rw-r--r--    1 root     root           919 Sep 29 04:28 mysite.log.1
  42870 -rw-r--r--    1 root     root           919 Sep 29 04:27 mysite.log.2
  43089 -rwxr-xr-x    1 root     root           233 Sep 26 04:03 mysite.uwsgi.ini
  43133 drwxr-xr-x    1 root     root          4096 Sep 26 04:47 polls
  32999 -rwxr-xr-x    1 root     root            14 Apr  3 15:46 requirements.txt
  42935 drwxr-xr-x    4 root     root          4096 Sep 26 04:47 static
```

Let's stay in the console. Next, we will dive into uWSGI.

uWSGI Log Rotation

In our sample project, the uWSGI log rotation is set in the **mysite.uwsgi.ini** file. It looks as below. It indicates that if the file size exceeds 2048 bytes, it would rotate.

[uwsgi]

socket = /tmp/uwsgi/mysite.sock
module = mysite.wsgi
master = true
processes = 2
chmod-socket = 666
logger = file:logfile=/tmp/uwsgi.log,maxsize=2048
vacuum = true

To list the uWSGI logs, type the following commands.

```
ls -li /tmp/
```

Refresh the web page a few times. A snapshot of the output is as below. Note that the log file whose inode is 42871 has been renamed while the file names of the rest of the copies stay the same. Since there is no limitation on the maximum number of copies, it would keep generating logs till the disk is full. If you know how to set the maximum number of copies, please leave your comment. It would be appreciated it. In this case, we can create a Cron job that cleans up old copies then.

```
/code # ls -li /tmp/
total 300
  27476 -rw-------    1 root     root        285222 Sep  8 00:46 tmpyi4hmf24
  74262 drwxr-xr-x    2 root     root          4096 Sep 29 03:51 uwsgi
  42871 -rw-r-----    1 root     root          1482 Sep 29 05:29 uwsgi.log
  42023 -rw-r-----    1 root     root          2053 Sep 29 03:59 uwsgi.log.1664423949
  42866 -rw-r-----    1 root     root          2073 Sep 29 04:13 uwsgi.log.1664424789
  42869 -rw-r-----    1 root     root          2071 Sep 29 04:28 uwsgi.log.1664425685
/code # ls -li /tmp/
total 304
  27476 -rw-------    1 root     root        285222 Sep  8 00:46 tmpyi4hmf24
  74262 drwxr-xr-x    2 root     root          4096 Sep 29 03:51 uwsgi
  42870 -rw-r-----    1 root     root           550 Sep 29 05:30 uwsgi.log
  42023 -rw-r-----    1 root     root          2053 Sep 29 03:59 uwsgi.log.1664423949
  42866 -rw-r-----    1 root     root          2073 Sep 29 04:13 uwsgi.log.1664424789
  42869 -rw-r-----    1 root     root          2071 Sep 29 04:28 uwsgi.log.1664425685
  42871 -rw-r-----    1 root     root          2050 Sep 29 05:30 uwsgi.log.1664429405
```

Linux logrotate Utility

Logrotate is a system utility that allows automatic rotation, compression, removal, and mailing of log files. A snapshot of the configuration is as below. It means the logrotate utility runs every day, doesn't write an error message if the log file is missing, keeps 14 old log files, compresses the rotated files, and prerotate/postrotate scripts only run once.

```
/var/log/uwsgi/*.log {
    daily
    missingok
    rotate 14
    compress
    sharedscripts
}
```

For the people who have experience with the very old version of uWSGI, the setup above might look familiar to you. We mentioned previously that uWSGI supports file-size-based rotation. Why do we need the logrotate utility to handle the rotation for the uWSGI log file? The reason is that back then, the old uWSGI server does not support file-size-based rotation. Thus, a workaround is to use logrotate as an alternative.

However, the above configuration has a problem. The uWSGI would keep writing to the same log file (same inode) even though the file has been renamed. To solve this issue, we must restart uWSGI after each rotation. As uWSGI restarts, it should generate a new log file (new inode). The revised configuration is as follows:

```
/var/log/uwsgi/*.log {
    daily
    missingok
    rotate 14
    compress
    sharedscripts
    postrotate
        systemctl stop uwsgi.service >/dev/null 2>&1
        systemctl start uwsgi.service >/dev/null 2>&1
    endscript
}
```

Conclusion

In this article, we went through three kinds of log rotation. They are application log rotation, server log rotation, and system log rotation. The discussion on the Linux logrotate utility does not end. Because the revised configuration has a negative impact. The application would have a hiccup during log rotation. If the business requires the app to be 24/7, it might be an issue.I would like to leave it open to you. Please leave comments and let me know your thoughts. Thanks for reading. Stay tuned.